NPSCS-92-003

NAVAL POSTGRADUATE SCHOOL
Monterey, California

RATIONAL BEHAVIOR MODEL: A TRI-LEVEL MULTIPLE PARADIGM ARCHITECHURE FOR ROBOT VEHICLE CONTROL SOFTWARE

S.H. Kwak, R.B. McGhee, and T.E. Bihari

March 1992

Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School
Monterey, California 93943

FedDocs
D 208.14/2
NPS-CS-92-003

LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

NAVAL POSTGRADUATE SCHOOL

Monterey, California

REAR ADMIRAL R. W. WEST, JR.
Superintendent

HARRISON SHULL
Provost

This report was prepared for and funded by the Naval Postgraduate School.

Reproduction of all or part of this report is authorized.

This report was prepared by:

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPSCS-92-003	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School 6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School 8b. OFFICE SYMBOL (if applicable) NPS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER OM&N Direct Funding
8c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
RATIONAL BEHAVIOR MODEL: A TRI-LEVEL MULTIPLE PARADIGM ARCHITECHURE FOR ROBOT VEHCILE CONTROL SOFT-

12. PERSONAL AUTHOR(S)
S.H. Kwak, R.B McGhee, and T.E. Bihari

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
	FROM _____ TO _____	March 1992	34

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	ROBOT VEHICLE, CONTROL, SOFTWARE ARCHITECHURE CONTROL SOFTWARE, ASV, AUV.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Computer control of robot vehicles of operation in unstructured environment typically involves both symbolic reasoning and numerical computation. Based on earlier experiences of the authors and others, this paper proposes a three level architecture for control software for such vehicles. From the top down, the three levels are called the strategic level, the tactical level, and the execution level. This paper argues that a multiple paradigm multi-lingual approach facilitates the realization of such a scheme with logic programming, object-oriented/functional programming, and imperative programming respectively being used in the successive levels. Experiments are reported in which Prolog is used at the strategic level, Lisp or C++ at the tactical level, and Pascal or C at the execution level. Experience with control of a large six legged walking machine and an autonomous submarine has led the authors to the belief that the proposed software architecture greatly facilitates the development and utilization of such vehicles through the isolation of a concise operational doctrine expressed horn clause form at the strategic level of control.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Se-Hung Kwak	22b. TELEPHONE (Include Area Code) (408) 646-2168 / 22c. OFFICE SYMBOL CS/Kw

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

GRADUATE SCHOOL KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CALIFORNIA 93943-5002

Title: Rational Behavior Model: A Tri-Level Multiple Paradigm Architecture for Robot Vehicle Control Software

Authors: S. H. Kwak*, R. B. McGhee*, and T.E. Bihari**

Affiliation: *Department of Computer Science, Naval Postgraduate School
Monterey, CA 93923
**Adaptive Machine Technologies, Inc.
1218 Kinnear Road, Columbus, Ohio 43212

Phone: 408-646-2168, 408-646-2449

E-Mail: kwak@cs.nps.navy.mil

Abstract

Computer control of robot vehicles for operation in unstructured environments typically involves both symbolic reasoning and numerical computation. Based on earlier experiences of the authors and others, this paper proposes a three level architecture for control software for such vehicles. From the top down, the three levels are called the strategic level, the tactical level, and the execution level. This paper argues that a multiple paradigm multi-lingual approach facilitates the realization of such a scheme with logic programming, object-oriented/functional programming, and imperative programming respectively being used in the successive levels. Experiments are reported in which Prolog is used at the strategic level, Lisp or C++ at the tactical level, and Pascal or C at the execution level. Experience with control of a large six-legged walking machine and an autonomous submarine has led the authors to the belief that the proposed software architecture greatly facilitates the development and utilization of such vehicles through the isolation of a concise operational doctrine expressed in Horn clause form at the strategic level of control.

Rational Behavioral Model: A Tri-Level Multiple Paradigm Architecture for Robot Vehicle Control Software

S.H. Kwak*, R.B. McGhee*, and T.E. Bihari**

*Department of Computer Science, Naval Postgraduate School
Monterey, CA 93943

**Adaptive Machine Technologies, Inc.
1218 Kinnear Road, Columbus, Ohio 43212

I. Introduction

Increasingly, robotic vehicles are being designed for real applications in unstructured environments. In such applications, they may be required to cooperate or share workspace with humans or other robots, or interact with equipment that is potentially hazardous.

Some preliminary work has been directed at defining "correct behavior" metrics for intelligent robotic vehicles [1]. However, existing design methodologies do not allow the behavior of even moderately complex robotic vehicles operating in moderately complex environments to be "proven" correct. Furthermore, we know of no general testing methodology that will ensure, with high confidence, that a complex robotic vehicle will always behave in a manner satisfactory to its human superiors.

We are interested in developing intelligent mobile robots that we can be confident will behave "rationally" with respect to some standards of correct behavior. The definition of

"rational" behavior is application-dependent. For example, in a military setting, a rational robot might base its overall behavior on the following types of guidelines:

Subordination: Carry out a specific "direct order" from a superior exactly as requested, to best of your ability.

Conflict Resolution: If direct orders from several superiors conflict, follow the orders of the highest ranking superior.

Continuity and Repeatability: When given less specific "general directives" by superiors, choose similar specific behaviors for "essentially similar" situations, and identical behaviors for "essentially identical" situations.

Domain-Specific Standards: A robotic vehicle might follow guidelines dealing with motion. For example:

Economy of Motion: Minimize changes in speed and direction, and make such changes smoothly. (A person standing next to a robotic vehicle would like to be confident that the vehicle will not suddenly jump sideways, for example.)

A central issue in building confidence in the rational behavior of a robotic vehicle is the problem of abstraction as a means for dealing with complexity. While any decomposition of a complex software system by means of an abstraction hierarchy is to some degree arbitrary, some decomposition strategies may be more effective at building confidence than others.

Many decomposition strategies have been proposed [2]. One major distinction between these strategies relates to the way in which primitive behaviors are invoked. At one extreme, the biologically motivated "behaviorist" school of thought tends to favor subsumption architectures in which each behavior carries with it its own (implicit) world model and is invoked directly by sensory data (or by "virtual sensors" [3]). One difficulty with this approach is that when two or more behaviors are active, competition for vehicle resources may result. In the pure version of subsumption, competing behaviors are always mediated so that the "most urgent" needs of the vehicle are met and less important behaviors are subsumed [4]. In other less extreme behaviorist approaches, competition for resources is resolved by some form of voting or command fusion [3].

While subsumption architectures show promise, confidence in the "rationality" of a subsumption-based robot vehicle depends on understanding the implicit mapping of low-level "data-driven" behaviors to the fulfillment of high-level goals. This is an area that requires further research.

In the research described in this paper, we have chosen a top level control strategy based on avoidance of behavioral conflicts (rather than their resolution) by making use of the inherent ordering of rules imposed by the backward chaining (goal driven) control structure of Prolog [5,6]. This approach to the sequencing of rule evaluation is at the opposite extreme from the forward chaining (data driven) behavior of subsumption architectures in the respect that invocation of behaviors is accomplished by a rigid operational doctrine designed specifically to prevent competition for vehicle resources by concurrently executing behaviors. It is thus optimized for assimilation and centralized control rather than for immediacy and decentralized control [3].

It is for this reason that we have chosen the term Rational Behavioral Model (RBM) to characterized our approach. As will be seen in the examples to follow, the rationality of RBM systems lies in the development of appropriate doctrines based on human reasoning about effective robot resource allocation and behavior.

II. Rational Behavioral Model

The hierarchical RBM architecture for mobile robots contains three levels: the strategic level, the tactical level, and the execution level.

The strategic level is the top level in the hierarchy. The strategic level maintains the vehicle's operational doctrine. This doctrine describes, in high-level, human-understandable terms, the allowable, "rational" behavior of the vehicle.

The execution level is the bottom level in the hierarchy. The execution level is responsible for interaction with the vehicle's sensors and actuators. This may include execution of servo control loops and limited sensor data integration, for example.

The tactical level is the intermediate level in the hierarchy. The tactical level has several duties. It assimilates data from execution-level components into a high-level "vehicle state" description suitable for use by the strategic level. It takes general "directives" (behavior commands) from the strategic level and elaborates them into specific commands for the execution level.

We have been guided in our choice of computer languages for each abstraction level by current trends in software engineering [7,8]. From our experiments, we have concluded

that a logic programming paradigm is effective for the strategic level, an object-oriented paradigm is well suited to the tactical level, and that the execution level seems best served by an imperative programming style [9]. This is not surprising. The strategic level is concerned with "rules" of good behavior, and logic programming allows these rules to be stated clearly and concisely, in human- readable form. The tactical level is concerned primarily with maintaining and operating upon the "vehicle's state", including coordinating the actions of the various components that make up the vehicle. Therefore, the object-oriented paradigm's ability to represent interacting objects is very useful. Finally, the execution level is concerned mainly with following the commands (imperatives) of the higher levels.

Our design of the three-level RBM architecture has been influenced by other models. In particular, Saridis [10] proposes a three-level organization of software for "intelligent" control systems consisting of a top organizational level based on symbolic information processing, a bottom execution level concerned with control of individual actuators by means of numerical computation, and an intermediate coordination level to deal with the symbolic/numeric interface and translation problem. He then points out that finite state machines provide a useful formalism for representing and implementing the coordination level. Similar abstraction levels have been proposed by others [11,12,13].

Because of obvious parallels between the operational characteristics of mobile robots and the "behavior" of animals, this term has acquired a special significance with regard to robot control systems [4,14]. In this paper, following [14], we will recognize both primitive behaviors and complex behaviors. Briefly, in our logic programming realization of the strategic level, a primitive behavior is an action invoked by a single message passed to the tactical level, while a complex behavior is a logical composition of primitive behaviors. To state this relationship in another way, the strategic level can be represented as an and/or tree

[12,15] (possibly with logical quantifiers at each node) in which leaf nodes are primitive behaviors and all other nodes represent complex behaviors. We find Horn clauses as implemented in Prolog to be an especially effective language for the representation and traversal of and/or trees [6,16] and have adopted it as our preferred means of expressing top level control strategies.

We have applied the above concepts to the problem of high-level motion control for a six-legged walking machine, the Adaptive Suspension Vehicle (ASV) [3,5,17]. More recently, we have attempted to escalate our application of the RBM architecture to the control of an unmanned vehicle, the NPS Model II Autonomous Underwater Vehicle (AUV) [18,19,20]. In this case, not only the overall regulation of behaviors, but also the entire mission control function is implemented via logic programming. Details of both the ASV and AUV application of the RBM architecture are provided in the remainder of this paper.

III. Robotic Vehicle Models

The ASV is a large, six-legged vehicle designed for outdoor operation in rough terrain. It is shown in Figure 1. The vehicle weighs approximately 7000 lbs and is 14 feet long. The ASV requires a human operator to guide its overall body movement, but limb motion coordination is accomplished by an on-board Intel 80386-based multiprocessor [5,17]. The software system is hierarchically organized with a clear distinction being made among an individual leg control level, a leg motion coordination level, and a body motion planning level [5,17]. Because the ASV has an omni-directional motion capability, it exhibits the general maneuverability characteristics of a helicopter. This behavior is achieved by providing the operator with a joystick with three major motion axes for control of vehicle forward velocity, lateral velocity, and turning velocity, respectively. The vehicle control

computer accepts these commands and synthesizes a sequence of leg movements to produce the desired body behavior. Therefore, eighteen degrees of freedom constituted by the six legs, each of which has three independent hydraulically operated actuators, is automatically coordinated by three velocity commands from a human operator. In this task the ASV is assisted by information from an optical terrain scanner which provides a map of terrain elevation in the immediate vicinity of the vehicle [17], and by force and position feedback from each leg. The ASV has successfully demonstrated its operational capability in various environments by climbing up steeply sloped areas, towing heavy loads on soft soils, traversing densely vegetated areas with minimal environmental damage, and crossing man-made obstacle areas, such as railroads, while maintaining a smooth and constant body attitude.

The NPS Model II AUV is a small underwater vehicle designed to support student and faculty research on autonomous robot control issues [18,19,20]. It is shown in Figure 2. The vehicle weighs approximately 380 lbs and is 84 inches long. As can be seen, the vehicle has a rectangular cross-section and is furnished with four forward control surfaces and four aft control surfaces as well as four tunnel thrusters. These thrusters, combined with the two aft screws, provide the vehicle with active control of five degrees of freedom in a low speed hovering mode, with only the roll degree of freedom being passively controlled. When the vehicle is operated in its higher-speed transit mode, thrusters are not used and all six degrees of freedom are actively controlled using the aft main screws for propulsion and hydrodynamic forces on the control surfaces to achieve commanded rotational rates in roll, pitch, and yaw. All of the eight control surfaces, two aft screws, and four thrusters are controlled by electric motors. The AUV does not require a human operator to guide its movement. Instead this function is performed by a on-board vehicle control computer. Currently, a Gespac 68030 based computer is the vehicle's sole computing resource, but it is anticipated that this system will soon be upgraded to a multi-

processor configuration. The AUV is equipped with depth and speed sensors, a complete suite of inertial sensors (3 rate gyros, 3 accelerometers, vertical gyro, directional gyro, and flux-gate compass), and a sonar system for obstacle avoidance and bottom sounding. The latter system consists of four fixed-base pencil-beam sonar rangers mounted in a flooded fiberglass nose cone. One sonar beam looks forward, another downward, and the other two are aimed perpendicular to the right and left of the forward looking beam. This vehicle is currently operated in a swimming pool environment at the Naval Postgraduate School and provides valuable experimental data and operational experience on an ongoing basis.

IV. ASV Control Implementation

Control software for the experimental evaluation of the ASV was developed in the 1980's and was written mainly in Pascal [17]. Because ASV testing was completed in 1989, it has not been possible to experimentally determine the effectiveness of the RBM architecture using this machine. Therefore, we have experimented with the RBM on a simulation of the ASV [5,21]. This simulation was developed based on data from the actual ASV. With respect to the needs of our experiments, the behaviors of the simulation mirror those of the ASV quite well.

The overall block diagram of the ASV simulator is shown in Figure 3. Each box shown in Figure 3 is an object that is an instance of a Flavor (early version of a common lisp class [22]) with the exception of the Free Gait Coordinator which is written in Symbolics Prolog [23]. Thus, the strategic level of control software is contained entirely within this block. Like the physical ASV which has nine major parts, namely, a body, a vision sensor, a cab, and six legs, the simulation object, "ASV" has correspondingly nine component objects, "Body", "Vision Sensor", "Joystick", and "Leg1" through "Leg6". These nine objects are

linked to "ASV" through a part relation implemented by an appropriate function call at ASV instantiation. Each part has its sub-parts, and again is linked to them with a part relation. Differing from the nine major parts which have visible corresponding parts in the real ASV, the subparts of the simulation are not physically tangible, but are introduced because of their functionalities for the simulator.

Besides the part relation, this figure also shows the hierarchical control architecture linking the simulation objects. As noted above, the top box labeled "Free Gait Coordinator" represents the strategic level of control. This box utilizes primitive behaviors provided by the "ASV" object, and gives commands to invoke these behaviors. As the name suggests, the major concern of this strategic (predicate calculus) level is to coordinate leg stepping and lifting events based on free (non-periodic) gaits. Of course, the top box also controls the body movement. However, body movement control is minimal in this box because the body movement is logically significantly simpler than controlling the six legs if free gaits are chosen for operation over rough terrain. Thus, the major consideration of this program is limb motion coordination, and the tri-level control architecture is rigorously adopted for this purpose. Specifically, the "Leg1 Control Machine", "Leg1 Executor" and "Leg1 Contact Sensor" correspond to the execution level, and the rest of the leg parts correspond to the tactical level. In the ASV implementation, the execution level has six copies of functional blocks for the six legs, and each one includes three objects. The six objects performing an identical functionality are created from one single class (or prototype). For example, "Leg1 Control Machine" through "Leg6 Control Machine" are instantiated from a class called "Leg Control Machine". This approach is utilized whenever multiple copies of functionally identical objects are necessary.

The "Leg1 Executor" and the "Leg1 Contact Sensor" in the execution level of "Leg1" take care of physical movement and external world interactions of the "Leg1" object. They are

under the supervision of the "Leg1 Control Machine" and provide necessary hardware status information in return. The "Leg1 Control Machine" has internal leg cycle logic, and related timing and external event sequence constraints. The leg motion cycle relative to the body during forward body motion over level terrain and its seven constituent leg phases are shown in Figure 4. Note that the leg movement in the "Lift" and "Descent" phases is parallel to the direction of gravity in order to eliminate the possibility of striking obstacles with the side of the leg. Evidently, the movement of a leg at a given moment always depends on the current phase and the status of a physical leg, such as its position relative to the body. In this implementation, the current leg phase information is memorized in the "Leg1 Control Machine" as a state, and the timing and external event sequence constraints are used as state transition conditions. Therefore, a seven-state finite machine is chosen to implement the "Leg1 Control Machine" Its state diagram is drawn in Figure 5. In this state machine, fixed time intervals, which represent the timing constraints, internally govern state transitions of the four synchronous states, while external events terminate the three asynchronous states. External events which arrive out of time expected sequence are automatically ignored due to the adopted finite state machine approach. Thus, the operation of "Leg1" is robust.

The "Leg1 Plan Machine" is the key component of the tactical level of the ASV implementation. The "Leg1 Foothold Finder" and "Leg1 TKM Calculator" perform foothold search and temporal kinematic margin calculation for "Leg1" as well as the "ASV" object [5]. The objects related to the vehicle body are also members of the tactical level, and support leg coordination through the "ASV" object. The "Leg1 Plan Machine" reports the availability and the current status of "Leg1" to the "ASV" through the "Leg1" object so that the "Free Gait Motion Coordinator", which is the top level in the tri-level control architecture, can utilize "Leg1" and plan its leg movement. Basically, at the top level, the availability of a leg for placing or lifting is sufficient information. Based on this

information, the top level will issue a command to "Leg1", which is known as its primitive behavior to the top level. The state diagram of the "Leg1 Plan Machine" is shown in Figure 6. This finite state machine has six states and all of them are asynchronous states. Thus, all the state transitions are controlled by external events. Specifically, the leg plan machines allow the top level planner to model the six legs with only two states, on the air and on ground, and to utilize three simple primitive behaviors; i.e., "Place", "Lift", and "Exchange". The "Exchange" command is introduced to improve performance on rough terrain, and basically causes the same action as a "Lift" command followed immediately by a "Place" command. The leg plan machines in the tactical level also isolate the top level planner from details of the physical legs, such as timing and physical constraints. When a leg placement action is requested by the planner, the leg plan machine replies positively as soon as the requested action is proved to be possible, even before the associated physical leg completes it. By doing this, the top level planner can plan ahead without waiting for completion of the requested actions. This also allows the planner to freely test possible leg placements and their consequences internally without concern for details or complicated prediction of the movement of physical legs. That is, for the planner, the six legs are massless two-state devices which can initiate a requested action immediately.

The "Leg1 Plan Machine" in the tactical level also provides an interface to the "Leg1 Control Machine" in the execution level. The given commands from the planner may not be immediately applicable depending on the current status of the corresponding physical leg. Specifically, the "Place" command from the top level planner can be immediately given to the lower level, but the actual touch-down event will not occur until the leg contacts the ground. On the other hand, a "Lift" command cannot be executed immediately either. This is due to the fact that the physical leg has to be in a legal state in order to execute the leg lifting action. Additionally, a leg lifting action is not allowed to violate the physical constraints of the ASV; i.e., stability must be maintained. Only when both of

these conditions are met does the "Leg1 Plan Machine" ask the "Leg1 Control Machine" to lift a leg by sending a "Recover_command". Consequently, the leg plan machines in the tactical level interface the top and bottom levels by making smooth transition from pure logical decisions to executable commands.

The internals of the "Free Gait Motion Coordinator" are shown in Figure 7. This code is written in Symbolics Prolog [23]. Differing from the lower levels, this level virtually does not need any further abstraction to explain its logic. This code itself is very readable by a person with an elementary knowledge of the Prolog syntax[1], and it is also executable by a computer without any modification. Basically, this code shows only a logical description of the whole system without showing implementation details in the lower levels. This code is started by typing "robot" from a computer terminal. This causes the program to execute the "initialize" clause and then to go into a loop which iteratively performs "get_command", "plan", and "execute". Because the ASV is driven by a human operator, the program gets commands from him by reading a joystick. After reading the joystick commands, it plans through execution of the "plan" clause. In the "plan" clause, there appears both "leg_plan" and "body_plan" clauses, which coordinate leg and body movement, respectively. When the program control reaches the "leg_plan" subgoal in the "plan" clause, the six "leg_plan" clauses are tested one by one from top to bottom until one of the clauses succeeds. Thus, the textual order presents logical priorities or preferences among them. This characteristic strictly comes from Prolog's conflict resolution strategy which utilizes depth first search. As can be seen in the code, the overall leg coordination strategy of this realization of free gait logic is based on minimizing number of legs on the ground. This is done to allow use of legs in the air to overcome unfavorable conditions frequently encountered from operation

[1] In reading this code, it is important to know that primitive behavior invocation and Lisp function calls are accomplished merely by executing a Prolog assignment statement; i.e., an is predicate call.

of the ASV on rough terrain. A more detailed explanation of this strategy and its effectiveness can be found in [5]. However, overall, it can be said that simulation results show that the strategy presented in Figure 7 gives better results than any other strategy investigated in crossing simulated rubble strewn terrain.

V. AUV Implementation

Though the control and physical architectural details of the AUV are totally different from the ASV, when the top level of AUV control software is compared with that of ASV, there exist many similarities between them. Practically, small differences come from the autonomous operation of the AUV. Specifically, referring to Figure 8, one top level predicate called "execute_auv_mission" starts the AUV operation, and this clause causes the system to be initialized and then to execute its main operation by repeating the "mission_control" subgoal. During initialization, first the system's integrity is checked, and then a mission scenario is downloaded from a mission planning expert system [18]. Currently, the mission scenario provided by the mission planner consists of a list of waypoints. Lastly, the first way point is selected as a first subgoal for the AUV to achieve along the way to its destination. The "mission_control" clause performs exactly the same function as the "loop" clause in the ASV. However, when the "get_command" subgoal is executed, a difference appears. Instead of getting joystick commands from a human operator, the commands are generated based on the current vehicle position and the current waypoint that the AUV is attempting to transit. Specifically, when the "get_waypoint" subgoal in the "get_command" clause is executed, two "get_waypoint" clauses are tested, one by one. If the current waypoint is reached, then a new waypoint is selected from the mission scenario by the first "get_waypoint" clause. Otherwise, no action (or default action) is initiated and control is passed to the "generate_command" subgoal. When this

clause is executed, vehicle heading and speed commands are generated without any intervention from a human operator. Consequently, the above clauses clearly show that the AUV is guided by waypoint control. The "plan" clauses also reflect unmanned operation. The first "plan" clause deals with unexpected situations, while the second "plan" clause involves executing a "normal_plan" subgoal. When the AUV encounters an uncharted obstacle, the down loaded mission scenario is modified by executing the "local_replan" subgoal so that the AUV can avoid an unexpected obstacle. In contrast, in the ASV, because a human operator constantly interacts with environment, there is no replanning at this level. Finally, it should be noted that those subgoals which are not refined further, such as "local_replan", "normal_plan", etc, reflect the incompleteness of the strategic level for the AUV at the time of this writing. Currently, this level as well as the two lower levels are under development.

VI. Summary and Discussion

As we have discussed, our goal is to develop mobile robots that can be counted upon to behave "rationally" from the perspective of the humans with which it must interact.

In the tri-level RBM control architecture, a global objective is clearly defined, and controls are centralized through a hierarchy among the levels. The strategic top level utilizes primitive behaviors provided by the tactical level and commands this level by sending a sequence of commands chosen based on its overall control strategy. The global control strategy, an abstracted operational doctrine, is given by a human who has developed it through his experience or observation of the target robot vehicle or a similar vehicle. Therefore, as long as this control strategy is correctly specified, the overall behavior of a robot vehicle is dependable and repetitive, as we usually expect from a man-made machine.

However, the control strategy cannot be obtained readily when a totally new robot vehicle with a different operational principle and functionality is designed, since the necessary human experience will be lacking. Nevertheless, if the strategic level of control logic for a similar robot vehicle is available, then the development effort for a new vehicle's top level will be significantly reduced. Thus, the experience obtained with the ASV significantly reduced development time for an initial version of the top level of control of the AUV.

The lower two levels of the tri-level control architecture are very closely tied to a physical architecture and an operational concept for a specific robot vehicle. For example, the ASV has six legs, and its two levels contain six six-state finite state machines and six seven-state finite state machines, respectively, with related supporting functionalities. The lower two levels of the AUV are under development, and a significantly different implementation is expected.

The basic concept of the tri-level control architecture is to some degree motivated by the physiology and anatomy of a human or an animal. Among the three levels, the top level mimics the function of our cerebrum in performing logic processing, and the functioning of the middle level resembles that of our cerebellum which interfaces and relays signals between our cerebrum and our limbs. Lastly, the lowest level interacts with motor and sensory devices somewhat like the human spinal reflex system. Despite these similarities, however, there are no self-learning and self-restructuring capabilities in the RBM architecture. Though those may be mandatory in a system that grows continuously like a biological system, this is not necessarily the case for a robot vehicle which could remain stable for a reasonable time before a major upgrade to its software. When there is a major change, all the levels should be updated accordingly. However, the proposed object implementation in this paper should significantly reduce the need for this kind of global upgrade. For example, when the original free gate simulator for the ASV was upgraded in

order to negotiate a long and narrow ditch in addition to randomly distributed terrain obstacles, the upgrade was completed with a minimal effort by introducing new behavioral functionalities, while inheriting the original behaviors in the original program code without any modification whatsoever [21].

The RBM Architecture depends upon multiple programming paradigms for its effective implementation; i.e., logic, object-oriented, functional, and imperative. However, while a multiple programming paradigm approach is logically clear and elegant, there can be a penalty with regard to computational efficiency. For example, Prolog does not match the hardware of typical microcomputers as well as conventional languages such as C, Ada, or C++. There is also overhead in interfacing multiple programming languages. Moreover, most hardware platforms and programming environments do not support all the above mentioned programming paradigms well. An alternative is to adopt multiple computing platforms and make each platform run its own programming paradigm.

Our application of the RBM to the AUV is proceeding. Under our current plan, the AUV will run the strategic level in Prolog on an Intel 80386 based computer and the tactical and execution levels in C++ and C on a Motorola 68030 based computer. A large part of the lower level software already has been completed and is currently being evaluated by frequent in-water testing of the AUV [24,25].

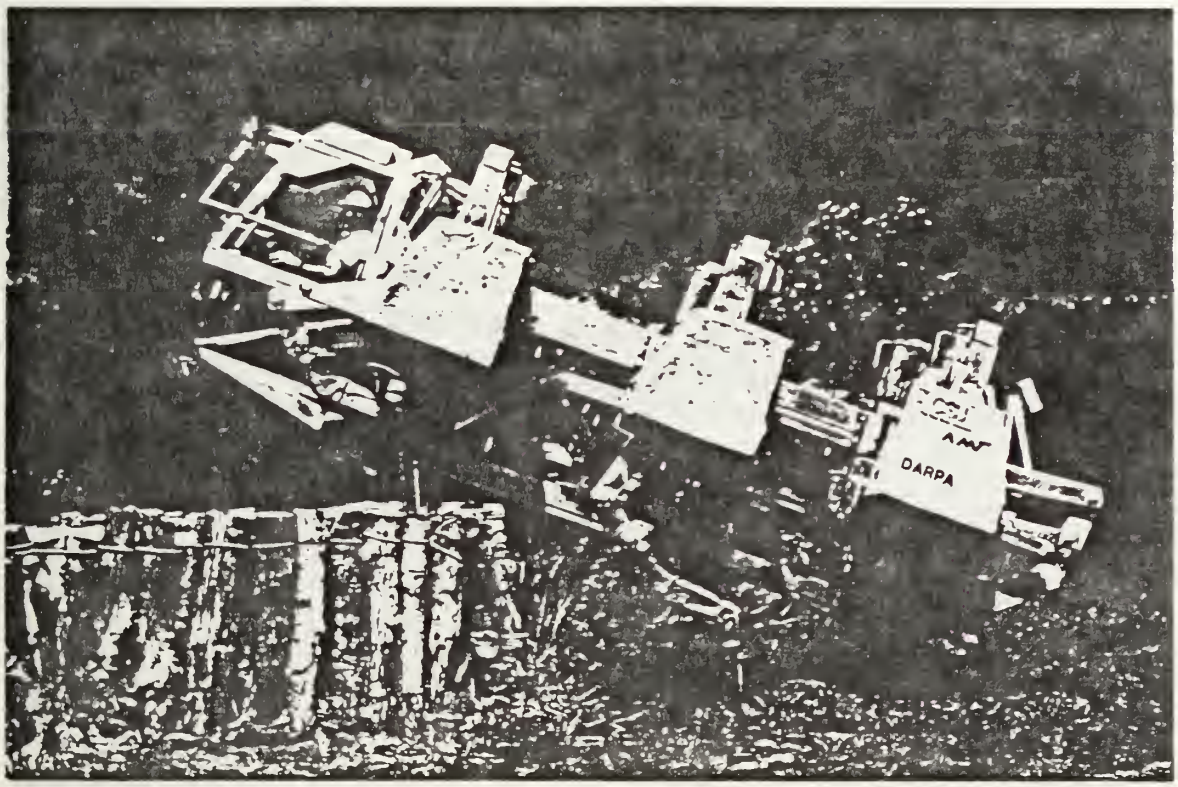

Figure 1: Adaptive Suspension Vehicle

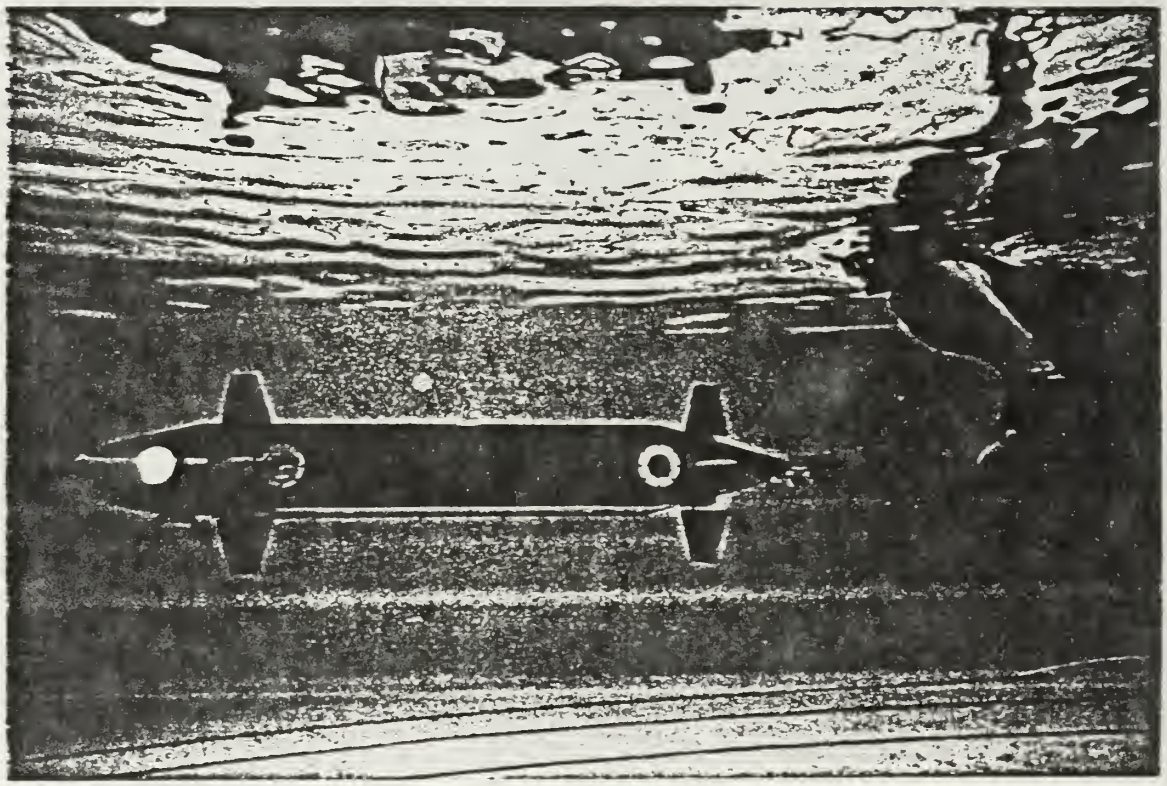

Figure 2: NPS Model II Autonomous Underwater Vehicle

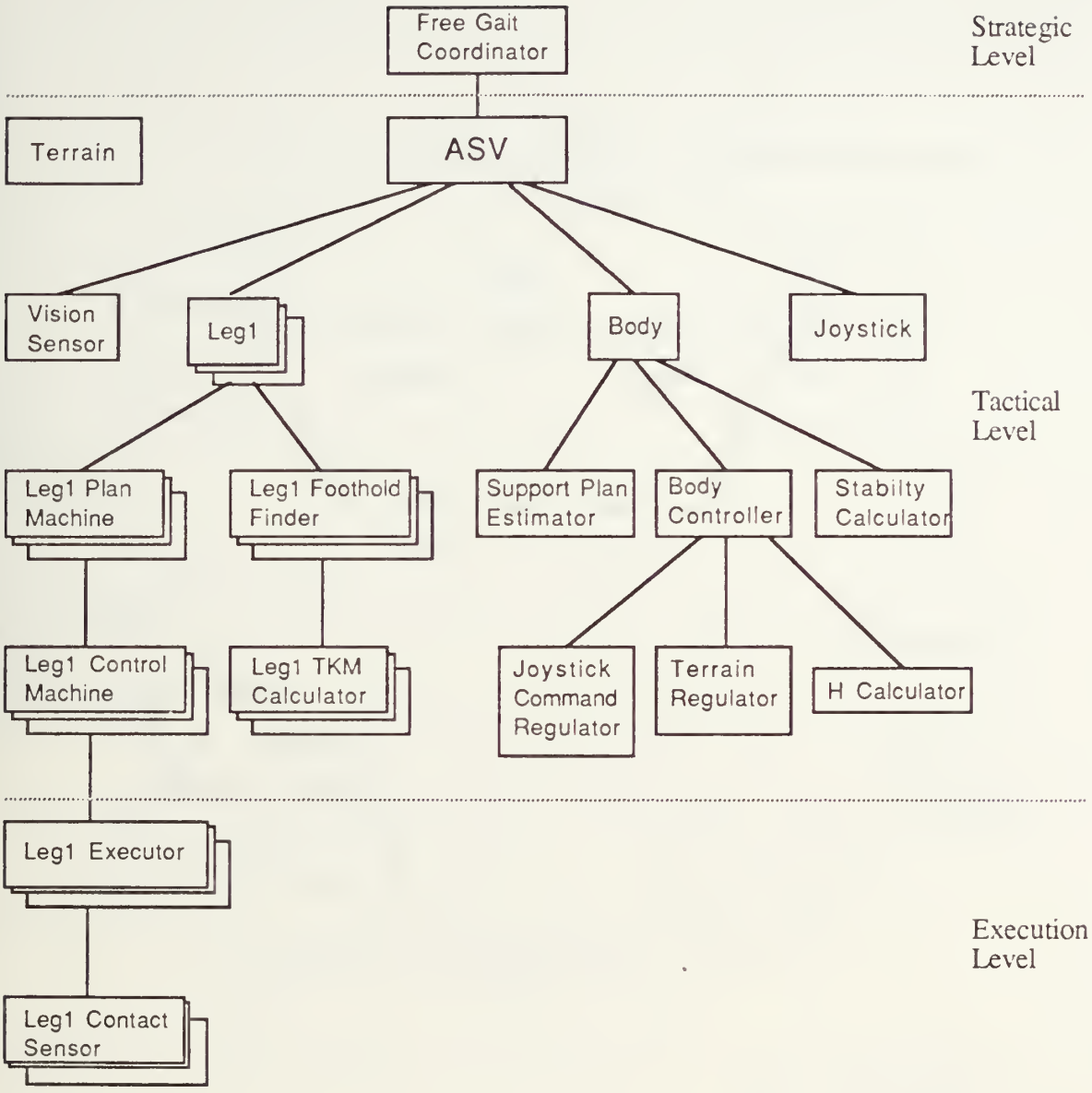

Figure 3: Hierarchy of ASV simulation objects

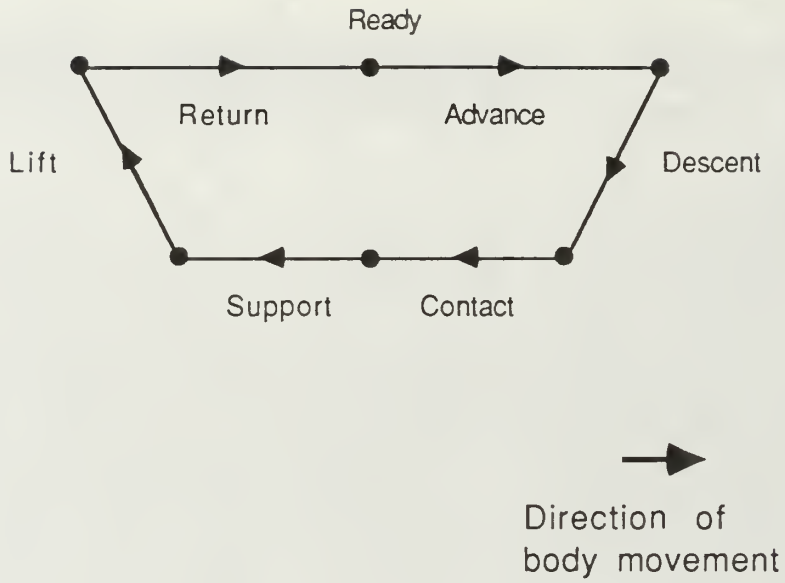

Figure 4: ASV leg motion relative to body during forward body motion over level terrain

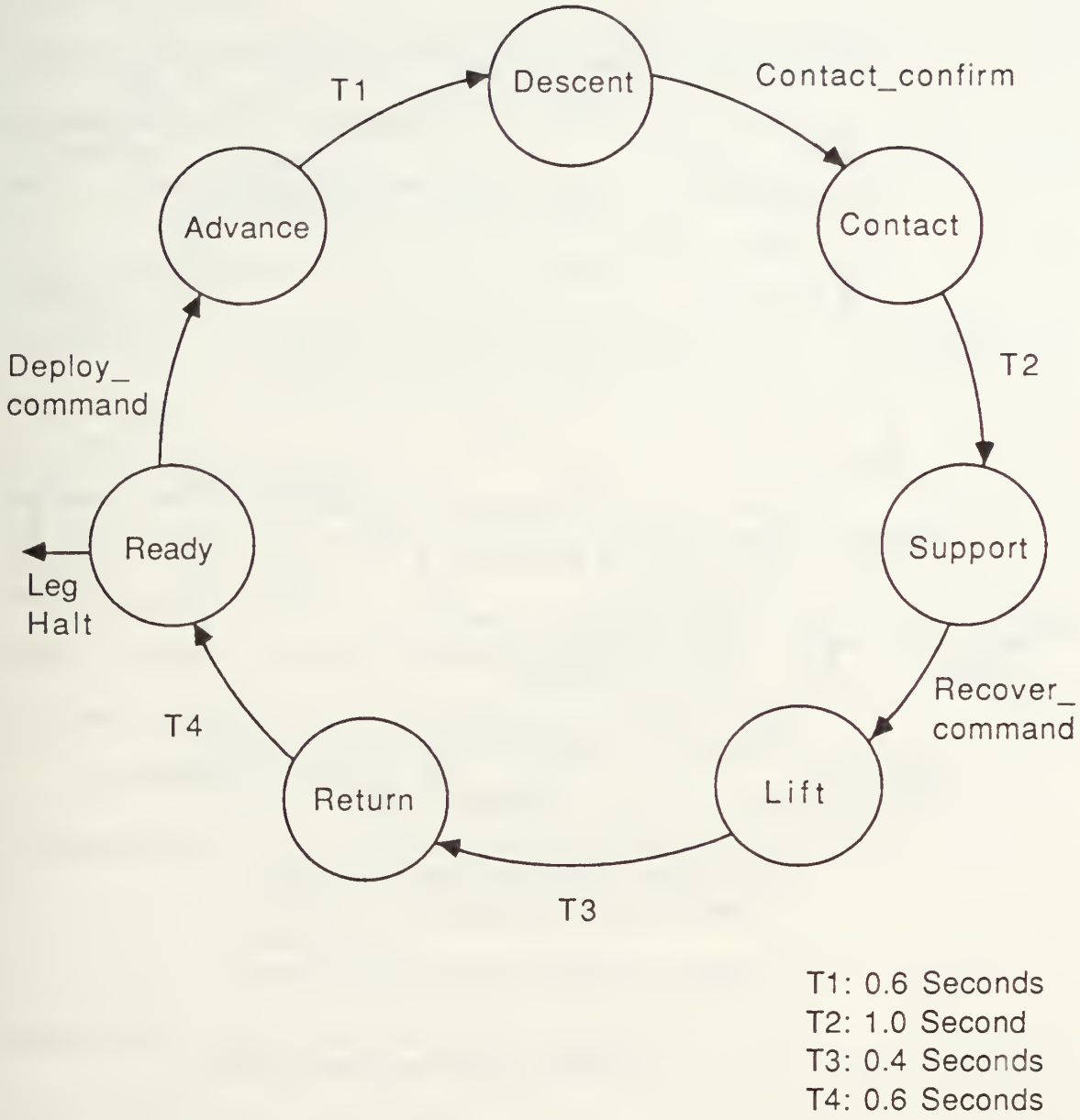

Figure 5: State diagram for ASV Leg Control Machine

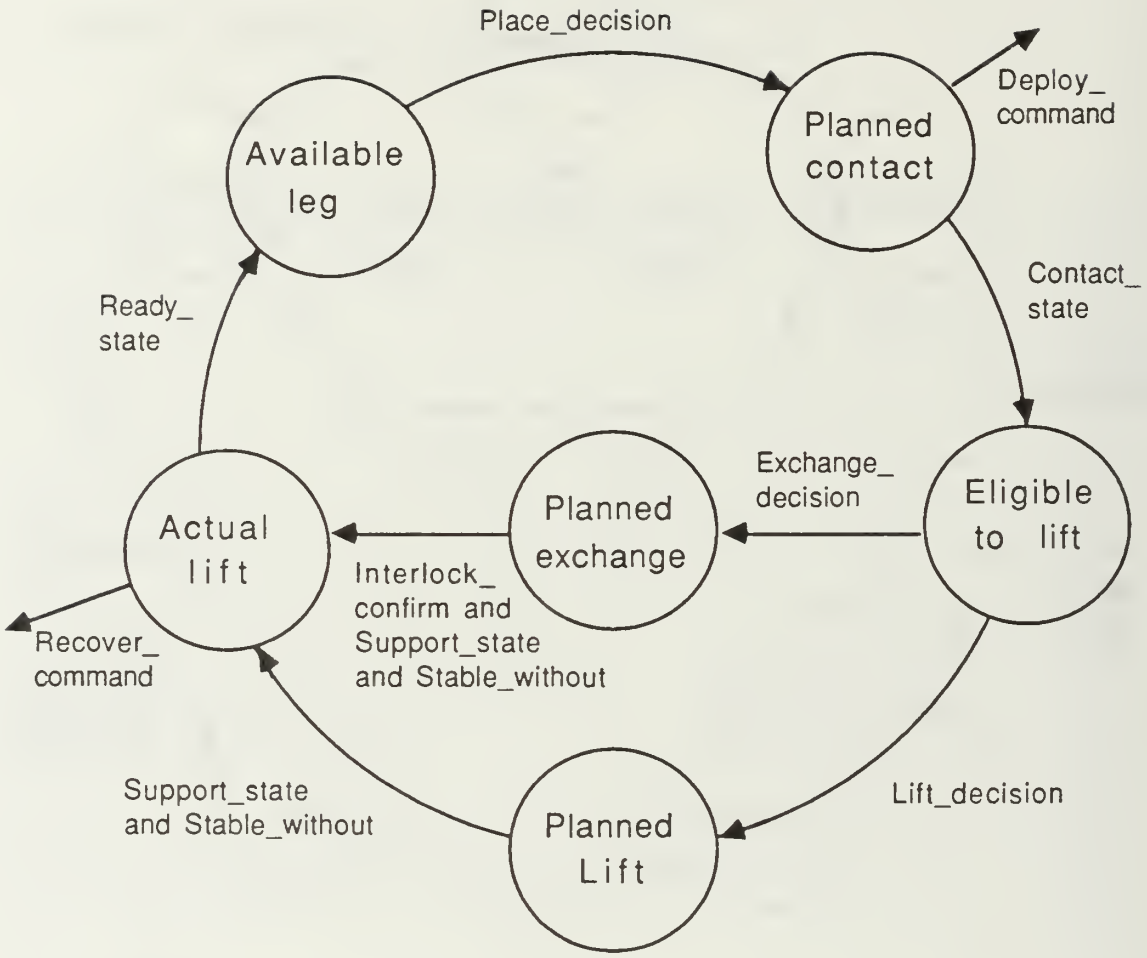

Figure 6: State diagram for ASV Leg Plan Machine

```
%%%  -*- Mode: PROLOG; Package: robot-rules -*-

robot :- initialize, repeat, loop, fail.

initialize :- X is inits.

loop :- get_command, plan, execute, !.

get_command :- X is read_joystick.

plan :- update_robot_state, check_tkm_limit,
        leg_plan, body_plan, generate_decision, !.

update_robot_state :- X is update_robot_status.

check_tkm_limit :- A_leg is at_tkm_limit, A_leg \== nil,
                   asserta(limit_leg(A_leg,lift)).
check_tkm_limit.

leg_plan :- lift_a_leg.
leg_plan :- exchange_legs.
leg_plan :- stable.
leg_plan :- place_a_leg.
leg_plan :- wait_for_legs.

stable :- Condition is stable_p, Condition == t.

lift_a_leg :- stable, A_leg is smallest_tkm_leg, A_leg \== nil,
              Condition is stable_without(A_leg), Condition == t,
              asserta(decision(A_leg,_,lift)).

exchange_legs :- stable, LegA is smallest_tkm_leg, LegA \== nil,
                 LegB is max_sm_leg(LegA), LegB \== nil,
                 Condition is has_more_tkm(LegB,LegA),
                 Condition == t,
                 asserta(decision(LegA,LegB,exchange)).

place_a_leg :- A_leg is max_sm_leg(_), A_leg \== nil,
               asserta(decision(A_leg,_,place)).

wait_for_legs :- try_new_foothold.
wait_for_legs :- recovery, asserta(reduce_speed).
wait_for_legs :- asserta(reduce_speed), restore_limit_leg.
```

Figure 7-1 : Free Gait Coordinator

```
try_new_foothold :- A_leg is leg_with_new_foothold, A_leg \== nil,
                    asserta(decision(A_leg,_,place)).

recovery :- A_leg is do_recovery, A_leg \== nil,
            asserta(decision(A_leg,_,place)), restore_limit_leg.

restore_limit_leg :- retract(limit_leg(A_leg,lift)).
restore_limit_leg.


body_plan :- speed_plan, trajectory_plan.

speed_plan :- retract(reduce_speed), slow_down.
speed_plan :- speed_up.

speed_up :- X is speed_up_robot.

slow_down :- X is slow_down_robot.

trajectory_plan :- stable_m, restore_trajectory.
trajectory_plan :- modify_trajectory.

stable_m :- Condition is stable_p_m, Condition == t.

restore_trajectory :- X is restore_command.

modify_trajectory :- X is modify_command.

generate_decision :- retract(decision(A_leg,B_leg,A_decision)),
                     X is send_decision(A_leg,B_leg,A_decision), fail.
generate_decision :- retract(limit_leg(A_leg,A_decision)),
                     X is send_decision(A_leg,_,A_decision), fail.
generate_decision.


execute :- execute_motion, draw_robot, !.

execute_motion :- X is execute_planned_motion.

draw_robot :- X is graphical_display.
```

Figure 7-2 : Continued.

```
execute_auv_mission :- initialize, repeat, mission_control, mission_completed.

initialize :- system_ok, download_mission, select_first_waypoint.

mission_control :- get_command, plan, execute_plan, !.

get_command :- get_waypoint, generate_command.

plan :- near_uncharted_obstacle, local_replan.
plan :- normal_plan.

mission_completed :- X is at_goal_point, X \== nil.

get_waypoint :- X is reach_waypoint, Y is get_next_waypoint.
get_waypoint.
```

Figure 8: AUV Mission Coordinator

References

[1] Brooks, R., "Planning and Control: Report from the Working Group", DARPA Winter Park Meeting, JUGVP, January, 1991.

[2] "Architectures for Real-Time Intelligent Control Systems: Concepts, Approaches, Methodologies", Record of the Architectural Approaches Working Group at the ARTICS-II Workshop.

[3] Payton, D.W., and Bihari, T.E., "Intelligent Real-Time Control of Robotic Vehicles", Communications of the ACM, Vol 34, No.8, August, 1991.

[4] Brooks, R., "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, pp. 14-23, 1986.

[5] Kwak, S.H., and McGhee, R.B., "Rule-based motion coordination for a hexapod walking machine", Advanced Robotics, Vol.4, No. 3, 1990, pp. 263-282.

[6] Rowe, N.C., Artificial Intelligence Through Prolog, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[7] Berzins, V., and Luqi, Software Engineering with Abstractions, Addision Wesley Publishing Company, Inc., Reading, Mass, 1991.

[8] Booch, G., Object Oriented Design with Applications, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.

[9] MacLennan, B.J., Principles of Programming Languages: Design, Evaluation, and Implementation, Holt, Rinehart and Winston, New York, New York, 1983.

[10] Saridis, G.N., "Intelligent Robotic Control", IEEE Transactions on Automatic Control, Vol. AC-28, No. 5, May 1983, pp. 547-556.

[11] McGhee, R.B., "Walking Machines", Ch.6 in Intelligent Systems, ed. by Hayes, J.E., and Michie, D., John Wiley and Sons, New York, 1983.

[12] Noreils, F.R., and Prajoux, R., "From Planning to Execution Monitoring Control for Indoor Mobile Robots", Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April, 1991, pp.1510-1517.

[13] Robinson, J., and Desrochers, A.A., "Performance Analysis of a Robotic Testbed Control Architecture", Proceedings of the 1990 IEEE International Conference on Robotics and Automation, Cincinnati, Ohio, 1990, pp.1782-1787.

[14] Arkin, R.C., and Gardner, W.F., "Reactive Inclinometer-based Mobile Robot Navigation", Proceedings of the 1990 IEEE International Conference on Robotics and Automation, Cincinnati, Ohio, 1990, pp. 936-941.

[15] Homem de Mello, L.S., and Sanderson, A.C., "Evaluation and Selection of Assembly Plans", Proceedings of the 1990 IEEE International Conference on Robotics and Automation, Cincinnati, Ohio, 1990, pp. 1588-1593.

[16] Missiaen, L., "Localized Abductive Planning for Robot Assembly", Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, April, 1991, pp. 605-610.

[17] Bihari, T. E., Walliser, T. M., and Patterson, M. R., "Controlling the Adaptive Suspension Vehicle," IEEE Computer Magazine, Vol. 22, No. 6, pp. 59-65, June 1989.

[18] Kwak, S.H., Ong, S.M., and McGhee., R.B., "A Mission Planning Expert System for an Autonomous Underwater Vehicle", Proceedings of the Symposium on Autonomous Underwater Vehicle Technology, IEEE Oceanic Engineering Society, June 4-6, 1990, Washington DC., pp. 123-128.

[19] Zyda, M.J., McGhee, R.B., Kwak, S.H., Nordman, D.B., Rogers, R.C, and Marco, D., "Three-Dimensional Visualization of Mission Planning and Control For the NPS Autonomous Underwater Vehicle", IEEE Journal of Oceanic Engineering, Vol, 15, No. 3, July, 1990.

[20] Healey, A.J., McGhee, R.B., Cristi, R., Papoulias, F.A., Kwak, S.H., Kanayama, Y., and Lee, Y., "Mission Planning, Execution, and Data Analysis for the NPS AUV II Autonomous Underwater Vehicle", Proc. of. 1st IARP Workshop on Mobile Robots for Subsea Environments, Monterey Bay Aquarium, Monterey, California, October 23-26, 1990, pp. 177-186.

[21] Kwak, S.H., and McGhee, R.B., Rule-Based Motion Coordination For The Adaptive Suspension Vehicle On Ternary-Type Terrain, Technical Report, No. NPSCS-91-006, Naval Postgraduate School, Monterey, California, December, 1990.

[22] Bromley, H., and Lamson, R., LIPS LORE: A Guide to Programming the LISP Machine", 2nd Ed., Kluwer Academic Publishers, Boston, 1987.

[23] Anon., User's Guide to Symbolics Prolog, Symbolics, Inc., Concord, MA, 1986.

[24] Papoulias, F.A., "Stability Considerations of Guidance and Control Laws for Autonomous Underwater Vehicles in the Horizontal Plane", Proc. of 7th International Symposium on Unmanned Untethered Submersible Technology, New England Center, Durham, New Hampshire, September 23-25, 1991.

[25] Floyd, C., Kanayama, Y., and Magrino, C., "Underwater Obstacle Recognition Using a Low-Resolution Sonar", Proc. of 7th International Symposium on Unmanned Untethered Submersible Technology, New England Center, Durham, New Hampshire, September 23-25, 1991.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314 — 2

Dudley Knox Library
Code 52
Naval Postgraduate School
Monterey, CA 93943-5100 — 2

Office of Research Administration
Code 08
Naval Postgraduate School
Monterey, CA 93943-5100 — 1

Chief of Naval Research
800 N. Quincy Street
Arlington, VA 22302-0268 — 1

Center for Naval Analyses
4401 Ford Avenue
Alexandria, VA 22302-0268

Chief of Naval Operations — 1
Director, Information Systems (OP-945)
Navy Department
Washington, D.C. 20350-2000

Chairman, Code CS — 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

Dr, Thomas E. Bihari — 1
Adaptive Machine Technologies, Inc.
1218 Kinnnear Road
Columbus, OH 43212

Prof. R.B. McGhee, Code CS/Mz — 3
Department of Computer Science
Naval postgraduate School
Monterey, CA 93943

Prof. Se-Hung Kwak, Code CS/Kw
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943

DUDLEY KNOX LIBRARY
3 2768 00347506 2